symfony a SENSIOLABS event

PARIS2009

11-12 Juin

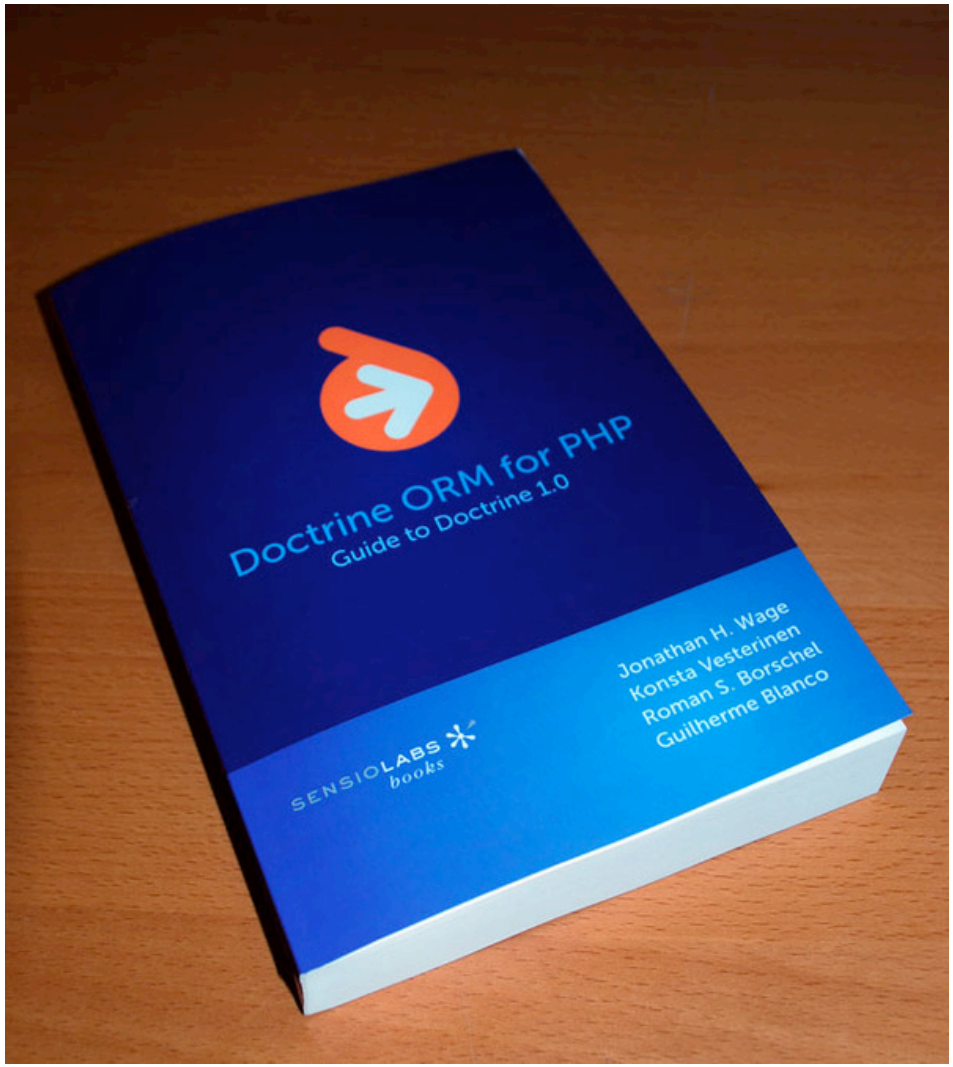
What's new in Doctrine

Jonathan H. Wage

What's new in Doctrine

- Doctrine Book
- Doctrine 1.1
- Doctrine 2.0
- Default ORM in Symfony as of today!!

The first Doctrine book



The first Doctrine book

- Available late June
- Available online(html/pdf) and in book format
- Complete user manual and reference guide for existing and new Doctrine developers

Doctrine 1.1

Evolution of 1.x codebase

Stability, bugs and minor features

Zero failing test cases

Dozens of new test cases adding more complete code coverage

Fine tuned API

Improved hydration performance

Hydrate larger result sets in less time

Doctrine 1.1

Re-written documentation which is also the book that will be available in print

New Features

New configuration options to remove hardcoded values

New Features

Specify more global default values

```php
$manager->setAttribute(
  Doctrine::ATTR_DEFAULT_TABLE_CHARSET,
  'utf8'
);
$manager->setAttribute(
  Doctrine::ATTR_DEFAULT_TABLE_COLLATE,
  'utf8_unicode_ci'
);
$manager->setAttribute(
  Doctrine::ATTR_DEFAULT_IDENTIFIER_OPTIONS,
  array('length' => 4)
);
$manager->setAttribute(
  Doctrine::ATTR_DEFAULT_COLUMN_OPTIONS,
  array('notnull' => true)
);
```

Global table character set

Global table collation

Default automatic identifier definition

Default column definition

Better custom accessor and mutator support

New Features

Custom accessors and mutators

```php
$manager->setAttribute(
  Doctrine::ATTR_AUTO_ACCESSOR_OVERRIDE,
  true
);

class User extends BaseUser
{
  public function setPassword($password)
  {
    $this->_set('password', md5($password));
  }
}

$user->password = 'changeme';
```

Feature is disabled by default

Define a new mutator

Set password property and setPassword() is invoked

New Features

Enhanced fromArray() and synchronizeWithArray() methods to better handle relationships

New Features

Define array structure

```
$userData = array(
  'username' => 'jwage',
  'password' => 'changeme',
  'Groups' => array(
    array(
      '_identifier' => 1,
    ),
    array(
      '_identifier' => 2
    ),
    array(
      'name' => 'New Group'
    )
  )
);
```

Use the array with an instance of a Doctrine model

```
$user = new User();
$user->fromArray($userData);
```

Internally Symfony uses these methods to merge form values to your model instances

Generate phpDoc property tags for integration with IDEs

New Features

All columns and relationships get a generated @property tag

```
/**
 * BaseUser
 *
 * This class has been auto-generated by the Doctrine ORM Framework
 *
 * @property string $username
 * @property string $password
 *
 * @package    ##PACKAGE##
 * @subpackage ##SUBPACKAGE##
 * @author     ##NAME## <##EMAIL##>
 * @version    SVN: $Id: Builder.php 5441 2009-01-30 22:58:43Z jwage $
 */
abstract class BaseUser extends Doctrine_Record
// ....
```

Offers auto complete for modern IDEs

Database Migrations

Database Migrations

General improvements
all around to make things more intuitive and flexible

Database Migrations

- Generated migration class files use timestamp prefix instead of incremented number

1.0	1.1
001_my_migration.class.php	1244735632_my_migration.class.php
002_my_migration2.class.php	1244735800_my_migration2.class.php

- Avoid conflicts between developers when generating migrations

Database Migrations

Introduced new Diff tool

Database Migrations

Generate migration classes automatically from changes made to your schema

Database Migrations

The schema we're migrating from

```
User:
  columns:
    username: string(255)
    password: string(255)
```

Database Migrations

The schema we're migrating to

```
User:
  columns:
    username: string(255)
    password: string(255)
    email_address: string(255)
```

We added a new email_address column

Database Migrations

- How can we get these changes to our DBMS in dev, staging, production, etc.?
  - Manually write the SQL and apply it in dev, staging and production? Eeeek!!
  - Make SQL change in dev server and filter all DDL statements from DBMS query log? Eeeek!!
- People handle this a lot of different ways and they are all dangerous and error prone
- Database migrations give you a reliable and programmatic way to deploy and even revert database changes

Database Migrations

Generating the changes

```php
$from = '/path/to/schema/from.yml';
$to = '/path/to/schema/to.yml';
$migrationsDir = '/path/to/migrations';
$diff = new Doctrine_Migration_Diff($from, $to, $migrationsDir);
$changes = $diff->generateChanges();

print_r($changes);
```

Database Migrations

The generated changes

```
Array
(
    [created_columns] => Array
        (
            [user] => Array
                (
                    [email_address] => Array
                        (
                            [type] => string
                            [length] => 255
                        )

                )

        )
)
```

Database Migrations

- Changes array used to generate migration classes
- The class generation is not 100%
- Some changes cannot be detected
  - For example we can't detect if a column was renamed or an old column dropped and a new one created.
- You should always review the generated classes to make sure they do what you want
- So now we can generate our classes to help us migrate our database changes

Database Migrations

Generating migration classes

```php
$from = '/path/to/schema/from.yml';
$to = '/path/to/schema/to.yml';
$migrationsDir = '/path/to/migrations';
$diff = new Doctrine_Migration_Diff($from, $to, $migrationsDir);
$diff->generateMigrationClasses();
```

New migration class written to the $migrationsDir

Database Migrations

Generating migration classes

```php
// /path/to/migrations/1239913213_version1.php

class Version1 extends Doctrine_Migration_Base
{
    public function up()
    {
        $this->addColumn('user', 'email_address', 'string', '255', array('email' => '1'));
    }

    public function down()
    {
        $this->removeColumn('user', 'email_address');
    }
}
```

Migrating changes

Migrate from version 0 to version 1

```
$migration = new Doctrine_Migration('migrations');
$migration->migrate();
```

Database Migrations

Migrating from version 0 to 1 would execute the up() methods

Database Migrations

Migrating from version 1 to 0 would execute the down() methods

Database Migrations

Migrating changes

Migrate back to version 0

```
$migration = new Doctrine_Migration('migrations');
$migration->migrate(0);
```

Database Migrations

Symfony CLI Workflow

- First modify your YAML schema
- Second run the following command

```
$ php symfony doctrine:generate-migrations-diff
```

- Now you need to review the generated migrations classes in lib/migration/doctrine. Once you confirm they look good run this command.

```
$ php symfony doctrine:migrate
```

- Your database is migrated and you can rebuild your models from your schema.

```
$ php symfony doctrine:build-model
```

Database Migrations

In Symfony migrations work by comparing your modified YAML schema to the old generated models

New Hydration Types

Scalar Hydration

Scalar Hydration

- Flat
- Rectangular result set
- Performs well
- Harder to work with
- Can contain duplicate data
- Like a normal SQL resultset

Scalar Hydration

```php
$q = Doctrine::getTable('User')
    ->createQuery('u')
    ->leftJoin('u.Phonenumbers p');

$results = $q->execute(array(), Doctrine::HYDRATE_SCALAR);

print_r($results);
```

Scalar Hydration

```
Array
(
    [0] => Array
        (
            [u_id] => 1
            [u_username] => jwage
            [u_password] => changeme
            [u_email_address] => jonwage@gmail.com
            [p_id] => 1
            [p_user_id] => 1
            [p_phonenumber] => 16155139185
        )

    [1] => Array
        (
            [u_id] => 1
            [u_username] => jwage
            [u_password] => changeme
            [u_email_address] => jonwage@gmail.com
            [p_id] => 2
            [p_user_id] => 1
            [p_phonenumber] => 14159925468
        )
)
```

Single Scalar Hydration

Single Scalar Hydration

Sub type of scalar hydration

Single Scalar Hydration

Returns single scalar value

Single Scalar Hydration

Useful for retrieving single value for aggregate/calculated results

Single Scalar Hydration

Very fast since no need exists to hydrate the data in to objects or any other structure

Single Scalar Hydration

```php
$q = Doctrine::getTable('User')
    ->createQuery('u')
    ->select('COUNT(p.id) as num_phonenumbers')
    ->leftJoin('u.Phonenumbers p');

$results = $q->execute(array(), Doctrine::HYDRATE_SINGLE_SCALAR);

echo $results; // 2
```

Doctrine 2.0

Requires PHP 5.3

Performance increase from 5.3

Doctrine 2.0

Test suite runs 20% faster and uses 30% less memory!

These performance increases are without any code changes. With Doctrine 1.x under 5.3 the same increases apply

Hydration Performance

Doctrine 1.1

4.3435637950897 for 5000 records

Doctrine 2.0

1.4314442552312 for 5000 records

Doctrine 2.0

3.4690098762512 for 10000 records

Everything re-designed and re-implemented

Simplified the public API

Heavily influenced by JPA/ Java Hibernate

JPA = Java Persistence API

Created by Sun

Smaller footprint

Un-necessary clutter removed

Removed Limitations

The old way

```php
class User extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('id', 'integer', null, array(
          'primary' => true,
          'auto_increment' => true
        ));

        $this->hasColumn('username', 'string', 255);
    }
}
```

Doctrine 2.0

The new way

```
/**
 * @DoctrineEntity
 * @DoctrineTable(name="user")
 */
class User
{
    /**
     * @DoctrineId
     * @DoctrineColumn(type="integer")
     * @DoctrineGeneratedValue(strategy="auto")
     */
    public $id;

    /**
     * @DoctrineColumn(type="varchar", length=255)
     */
    public $username;
}
```

No need to extend a base class anymore

No more cyclic references

print_r() your objects

```php
$user = new User();
$user->username = 'jwage';
print_r($user);
```

```
User Object
(
    [id] =>
    [username] => jwage
)
```

Doctrine 2.0

- Positive effect of removing the base class all around
- Performance increase
- Easier to debug

No more shared identity map across connections

Other general improvements

Code heavily de-coupled

Three main packages

Common

DBAL

Doctrine 2.0

ORM

Use just the DBAL without the ORM present

Database Abstraction Layer

Doctrine 2.0

- Under the hood of Doctrine is a powerful database abstraction layer.
- It is an evolution of code that has existed in other projects such as PEAR MDB/MDB2 and Zend_Db
- This layer has always existed but not advertised.
- Now that it can be used standalone we'll be advertising it much more as a separate system.

A few examples of the DBAL

Programatically issue DDL statements

```php
$columns = array(
    'id' => array(
        'type' => \Doctrine\DBAL\Type::getType('integer'),
        'autoincrement' => true,
        'primary' => true,
        'notnull' => true
    ),
    'test' => array(
        'type' =>  \Doctrine\DBAL\Type::getType('string'),
        'length' => 255
    )
);

$options = array();

$sm->createTable('new_table', $columns, $options);
```

Try a method. If an error occurs in the DBAL an exception is thrown

Sometimes you may want to catch the exception and swallow it.

Doctrine 2.0

Use the tryMethod() to.....try a method and return false if it fails. The exception that was thrown if any can be retrieved and inspected.

```php
if ($sm->tryMethod('createTable', 'new_table', $columns, $options)) {
  // do something
}
```

Doctrine 2.0

Often when issuing DDL statements you are creating something that may or may not exist already so you might want to get rid of it first before re-creating it.

Doctrine 2.0

Previously in Doctrine 1.x, the code for that might look something like the following for dropping and creating a database.

```
try {
  $sm->dropDatabase('test_db');
} catch (Exception $e) {}

$sm->createDatabase('test_db');
```

Doctrine 2.0

But now we have added new methods for dropping and creating things so the code looks like this now.

```
$sm->dropAndCreateDatabase('test_db');
```

Every create*() method has a matching dropAndCreate*() method

Doctrine 2.0

You'll be hearing a lot more about the Doctrine DBAL but for now we'll get back to talking about what else is new in Doctrine

Due to the decoupling things are easier to extend and override

Better support for multiple database connections

Sequences, schemas and catalogs fully supported

Simplified connection information

```php
$config = new \Doctrine\ORM\Configuration();
$eventManager = new \Doctrine\Common\EventManager();
$connectionOptions = array(
    'driver' => 'pdo_sqlite',
    'path' => 'database.sqlite'
);
$em = \Doctrine\ORM\EntityManager::create(
    $connectionOptions, $config, $eventManager
);
```

No more DSN nightmares

Connection information specified as simple PHP arrays

Removed old and heavy constant based attribute system

Replaced with a simpler and lighter string based configuration system

Real Native SQL support

Driver based Meta Data

Annotations

```php
/**
 * @DoctrineEntity
 * @DoctrineTable(name="user")
 */
class User
{
    /**
     * @DoctrineId
     * @DoctrineColumn(type="integer")
     * @DoctrineGeneratedValue(strategy="auto")
     */
    public $id;

    /**
     * @DoctrineColumn(type="varchar", length=255)
     */
    public $username;
}
```

PHP Code

```php
$metadata = new ClassMetadata('User');

$metadata->mapField(array(
  'fieldName' => 'id',
  'type' => 'integer',
  'id' => true
));

$metadata->setIdGeneratorType('auto');

$metadata->mapField(array(
  'fieldName' => 'username',
  'type' => 'varchar',
  'length' => 255
));
```

YAML

```yaml
User:
  properties:
    id:
      id: true
      type: integer
      idGenerator: auto
    username:
      type: varchar
      length: 255
```

Doctrine 2.0

- Other drivers possible
- XML, PHP arrays, etc.
- Write your own driver

Caching

Query Cache

Cache final SQL that is generated from parsing DQL

Metadata Cache

Cache the metadata containers so they are only populated once

Result Cache

Cache the result sets of your queries

Inheritance Mapping

Single Table

One table per hierarchy

Class Table

One table per class

Concrete Table

One table per concrete class

Testing

Switched to phpUnit

Better mock testing

Test suite can be ran against any DBMS. MySQL, PgSQL, Oracle, Sqlite, etc.

Because of code decoupling tests are more granular and easier to debug

New Features

New DQL Parser

Hand written recursive descent parser

Constructs AST objects

PHP class names of DQL parser directly represent the language itself

OrderByClause.php
SelectClause.php
SelectExpression.php
Subselect.php
DeleteClause.php
etc. etc.

Every DQL feature has a class to handle the parsing

Doctrine 2.0

- Easy to use
- Easy to expand and add new features
- Easy to use and understand the parsing of a DQL string
- Expand the DQL parser with your own functionality and add to the DQL language

Performance of DQL parser is irrelevant due to the parsing being cached

Custom Data Types

Doctrine 2.0

```php
namespace \Doctrine\DBAL\Types;

class MyCustomObjectType extends Type
{
    public function getName()
    {
        return 'MyCustomObjectType';
    }

    public function getSqlDeclaration(array $fieldDeclaration,
    \Doctrine\DBAL\Platforms\AbstractPlatform $platform)
    {
        return $platform->getClobDeclarationSql($fieldDeclaration);
    }

    public function convertToDatabaseValue($value,
    \Doctrine\DBAL\Platforms\AbstractPlatform $platform)
    {
        return serialize($value);
    }

    public function convertToPHPValue($value)
    {
        return unserialize($value);
    }
}
```

Add the custom type

```
Type::addCustomType(
  'MyCustomObjectType',
  'Doctrine\DBAL\Types\MyCustomObjectType'
);
```

Overriding data types

Override Types

```
class MyString extends StringType
{

}

Type::overrideType('string', 'Doctrine\DBAL\Types\MyString');
```

Questions

Jonathan H. Wage
jonathan.wage@sensio.com
+1 415 992 5468

sensiolabs.com | doctrine-project.org | sympalphp.org | jwage.com

You can contact Jonathan about Doctrine and Open-Source or for training, consulting, application development, or business related questions at jonathan.wage@sensio.com